# FOUNDATION-0.2d: Resource Schema - PART 1 (Code)

## 🎯 CONTEXT

**Phase:** FOUNDATION (Week 1 - Day 2 Morning)
**Component:** Resource Agent Database Schema
**Estimated Time:** 15 min AI execution + 15 min verification
**Complexity:** MEDIUM
**Risk Level:** LOW
**Files:** Part 1 of 2 (Code implementation)

---

## 📦 DEPENDENCIES

### Must Complete First:

- **FOUNDATION-0.2a:** Core Database Schema ✅ COMPLETED
  - All 6 core tables created
  - Tests passing
- **FOUNDATION-0.2b:** Agent State Tables ✅ COMPLETED
  - All 4 agent state tables created
  - Tests passing
- **FOUNDATION-0.2c:** Workflow History Tables ✅ COMPLETED
  - All 3 workflow history tables created
  - Tests passing

### Required Services Running:


bash

```
# Verify all services are healthy
cd ~/optiinfra
make verify

# Expected output:
# PostgreSQL... ✅ HEALTHY
# ClickHouse... ✅ HEALTHY
# Qdrant... ✅ HEALTHY
# Redis... ✅ HEALTHY
```

### Database State:

bash

```
# Verify existing tables
psql postgresql://optiinfra:password@localhost:5432/optiinfra -c "\dt"

# Expected: 13 tables from 0.2a + 0.2b + 0.2c
# customers, agents, events, recommendations, approvals, optimizations
# agent_configs, agent_states, agent_capabilities, agent_metrics
# workflow_executions, workflow_steps, workflow_artifacts
```

---

# 🎯 OBJECTIVE

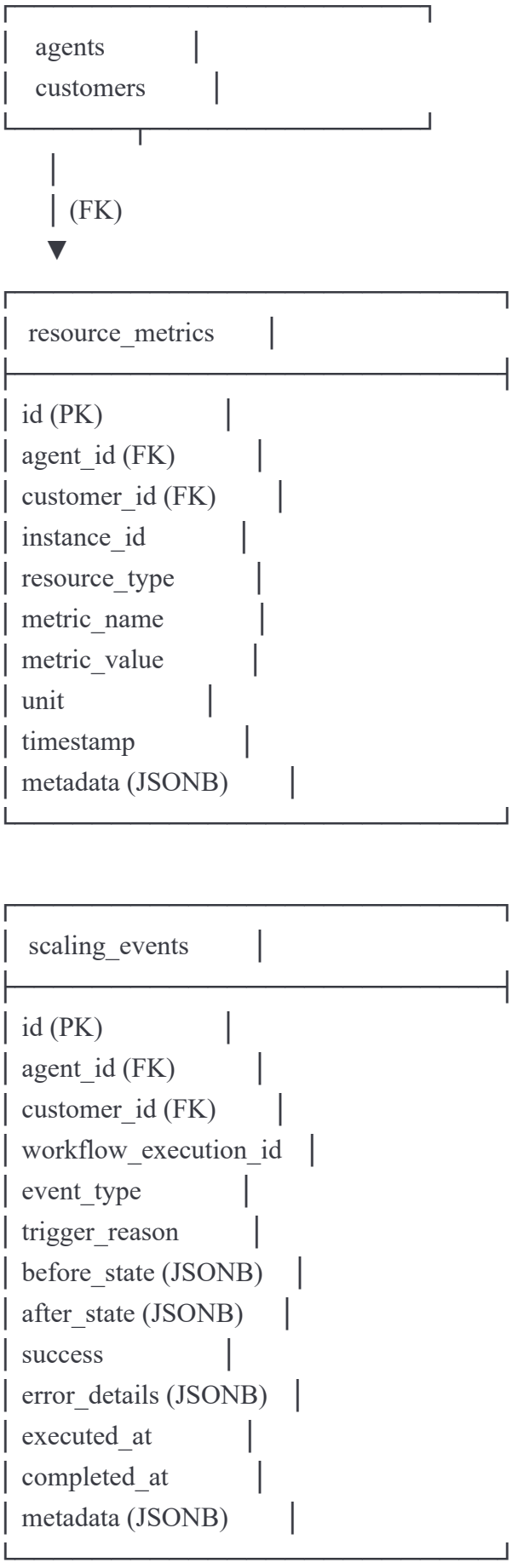
Add **Resource Agent-specific database tables** to track GPU/CPU/memory utilization and auto-scaling events. These tables enable:

1. **Resource Metrics** - Track GPU utilization, CPU usage, memory, disk, network
2. **Scaling Events** - Record all auto-scaling decisions and outcomes

### What This Enables:

- ✅ Track GPU utilization per instance over time
- ✅ Monitor CPU and memory usage patterns
- ✅ Record auto-scaling decisions (scale up/down)
- ✅ Analyze scaling effectiveness
- ✅ Link scaling to workflows and cost savings
- ✅ Historical resource utilization analysis
- ✅ Predictive scaling based on patterns

### Database Design:


agents
customers
(FK)
resource_metrics
id (PK)
agent_id (FK)
customer_id (FK)
instance_id
resource_type
metric_name
metric_value
unit
timestamp
metadata (JSONB)
scaling_events
id (PK)
agent_id (FK)
customer_id (FK)
workflow_execution_id
event_type
trigger_reason
before_state (JSONB)
after_state (JSONB)
success
error_details (JSONB)
executed_at
completed_at
metadata (JSONB)

# 📁 FILE 1: Resource Schema Models

**Location:** `~/optiinfra/shared/database/models/resource_schema.py`

python

```
"""
Resource Schema Models (FOUNDATION-0.2d)
Tracks resource metrics and auto-scaling events
"""

from datetime import datetime
from sqlalchemy import (
    Column, String, DateTime, Float, Boolean, Enum as SQLEnum,
    ForeignKey, Index
)
from sqlalchemy.dialects.postgresql import UUID, JSONB
from sqlalchemy.orm import relationship
import uuid
import enum

from shared.database.base import Base


# ============================================
# ENUMS
# ============================================

class ResourceType(str, enum.Enum):
    """Types of resources being monitored"""
    GPU = "gpu"
    CPU = "cpu"
    MEMORY = "memory"
    DISK = "disk"
    NETWORK = "network"


class ScalingEventType(str, enum.Enum):
    """Types of scaling events"""
    SCALE_UP = "scale_up"
    SCALE_DOWN = "scale_down"
    AUTO_SCALE_TRIGGERED = "auto_scale_triggered"
    MANUAL_SCALE = "manual_scale"
    SCALE_CANCELLED = "scale_cancelled"


# ============================================
```

```
# MODELS
# ============================================

class ResourceMetric(Base):
    """
    Tracks resource utilization metrics

    Stores GPU utilization, CPU usage, memory, disk I/O, network bandwidth
    for each instance. Supports time-series analysis of resource usage.
    """
    __tablename__ = "resource_metrics"

    # Primary Key
    id = Column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4,
        index=True
    )

    # Foreign Keys
    agent_id = Column(
        UUID(as_uuid=True),
        ForeignKey("agents.id", ondelete="CASCADE"),
        nullable=False,
        index=True
    )
    customer_id = Column(
        UUID(as_uuid=True),
        ForeignKey("customers.id", ondelete="CASCADE"),
        nullable=False,
        index=True
    )

    # Instance Identification
    instance_id = Column(
        String(255),
        nullable=False,
        index=True,
        comment="Cloud instance ID (e.g., i-0abc123, instance-xyz)"
    )
```

```
    # Resource Details
    resource_type = Column(
        SQLEnum(ResourceType, name="resource_type"),
        nullable=False,
        index=True
    )
    metric_name = Column(
        String(100),
        nullable=False,
        index=True,
        comment="Metric name: utilization, temperature, memory_used, etc."
    )
    metric_value = Column(
        Float,
        nullable=False,
        comment="Numeric value of the metric"
    )
    unit = Column(
        String(50),
        nullable=False,
        comment="Unit: percent, celsius, MB, GB, etc."
    )

    # Timing
    timestamp = Column(
        DateTime,
        nullable=False,
        index=True,
        comment="When the metric was collected"
    )

    # Additional Data
    metadata = Column(
        JSONB,
        nullable=False,
        default=dict,
        comment="Additional context: gpu_model, instance_type, etc."
    )

    # Audit
```

```
    created_at = Column(DateTime, default=datetime.utcnow, nullable=False)

    # Relationships
    agent = relationship("Agent", back_populates="resource_metrics")
    customer = relationship("Customer", back_populates="resource_metrics")

    # Indexes
    __table_args__ = (
        Index(
            "ix_resource_metrics_agent_instance",
            "agent_id", "instance_id"
        ),
        Index(
            "ix_resource_metrics_customer_type",
            "customer_id", "resource_type"
        ),
        Index(
            "ix_resource_metrics_timestamp",
            "timestamp"
        ),
        Index(
            "ix_resource_metrics_instance_timestamp",
            "instance_id", "timestamp"
        ),
        Index(
            "ix_resource_metrics_type_name",
            "resource_type", "metric_name"
        ),
    )

    def __repr__(self):
        return (
            f"<ResourceMetric(id={self.id}, "
            f"instance={self.instance_id}, type={self.resource_type}, "
            f"metric={self.metric_name}, value={self.metric_value})>"
        )


class ScalingEvent(Base):
    """
    Tracks auto-scaling events
```

```
Records all scaling decisions (up/down), their triggers, outcomes,
and effects on the system.
"""
__tablename__ = "scaling_events"

# Primary Key
id = Column(
    UUID(as_uuid=True),
    primary_key=True,
    default=uuid.uuid4,
    index=True
)

# Foreign Keys
agent_id = Column(
    UUID(as_uuid=True),
    ForeignKey("agents.id", ondelete="CASCADE"),
    nullable=False,
    index=True
)
customer_id = Column(
    UUID(as_uuid=True),
    ForeignKey("customers.id", ondelete="CASCADE"),
    nullable=False,
    index=True
)
workflow_execution_id = Column(
    UUID(as_uuid=True),
    ForeignKey("workflow_executions.id", ondelete="SET NULL"),
    nullable=True,
    index=True,
    comment="Link to workflow that triggered scaling"
)

# Event Details
event_type = Column(
    SQLEnum(ScalingEventType, name="scaling_event_type"),
    nullable=False,
    index=True
)
```

```
    trigger_reason = Column(
        String(500),
        nullable=False,
        comment="Why scaling was triggered: high_gpu_util, low_cpu_util, etc."
    )

    # State Changes
    before_state = Column(
        JSONB,
        nullable=False,
        comment="State before scaling: instance_count, avg_utilization, etc."
    )
    after_state = Column(
        JSONB,
        nullable=False,
        comment="State after scaling: new_instance_count, new_utilization, etc."
    )

    # Outcome
    success = Column(
        Boolean,
        nullable=False,
        default=False,
        index=True
    )
    error_details = Column(
        JSONB,
        nullable=True,
        comment="Error details if scaling failed"
    )

    # Timing
    executed_at = Column(
        DateTime,
        nullable=False,
        index=True,
        comment="When scaling was initiated"
    )
    completed_at = Column(
        DateTime,
        nullable=True,
```

```
    comment="When scaling completed (or failed)"
)

# Additional Data
metadata = Column(
    JSONB,
    nullable=False,
    default=dict,
    comment="Additional context: cost_impact, performance_impact, etc."
)

# Audit
created_at = Column(DateTime, default=datetime.utcnow, nullable=False)
updated_at = Column(
    DateTime,
    default=datetime.utcnow,
    onupdate=datetime.utcnow,
    nullable=False
)

# Relationships
agent = relationship("Agent", back_populates="scaling_events")
customer = relationship("Customer", back_populates="scaling_events")
workflow = relationship(
    "WorkflowExecution",
    foreign_keys=[workflow_execution_id],
    backref="scaling_events"
)

# Indexes
__table_args__ = (
    Index(
        "ix_scaling_events_agent_type",
        "agent_id", "event_type"
    ),
    Index(
        "ix_scaling_events_customer_success",
        "customer_id", "success"
    ),
    Index(
        "ix_scaling_events_executed_at",
```

```
            "executed_at"
        ),
        Index(
            "ix_scaling_events_workflow",
            "workflow_execution_id"
        ),
    )

    def __repr__(self):
        return (
            f"<ScalingEvent(id={self.id}, type={self.event_type}, "
            f"success={self.success}, executed_at={self.executed_at})>"
        )

    @property
    def duration_seconds(self):
        """Calculate scaling duration in seconds"""
        if self.completed_at and self.executed_at:
            return (self.completed_at - self.executed_at).total_seconds()
        return None
```

---

# 📁 FILE 2: Update Core Models (Add Relationships)

**Location:** `~/optiinfra/shared/database/models/core.py`

**Add these relationships to existing models:**

python

```python
# In the Agent model, add:
from sqlalchemy.orm import relationship

class Agent(Base):
    # ... existing fields ...

    # ADD THESE NEW RELATIONSHIPS:
    resource_metrics = relationship(
        "ResourceMetric",
        back_populates="agent",
        cascade="all, delete-orphan"
    )
    scaling_events = relationship(
        "ScalingEvent",
        back_populates="agent",
        cascade="all, delete-orphan"
    )


# In the Customer model, add:
class Customer(Base):
    # ... existing fields ...

    # ADD THESE NEW RELATIONSHIPS:
    resource_metrics = relationship(
        "ResourceMetric",
        back_populates="customer",
        cascade="all, delete-orphan"
    )
    scaling_events = relationship(
        "ScalingEvent",
        back_populates="customer",
        cascade="all, delete-orphan"
    )
```

---

# 📁 FILE 3: Update Model Imports

**Location:** `~/optiinfra/shared/database/models/__init__.py`

python

```
"""
Database models package
"""

# Core models (0.2a)
from shared.database.models.core import (
    Customer,
    Agent,
    Event,
    Recommendation,
    Approval,
    Optimization,
    # Enums
    CustomerStatus,
    AgentType,
    AgentStatus,
    EventType,
    EventSeverity,
    RecommendationType,
    ApprovalStatus,
    OptimizationStatus,
)

# Agent state models (0.2b)
from shared.database.models.agent_state import (
    AgentConfig,
    AgentState,
    AgentCapability,
    AgentMetric,
    # Enums
    ConfigType,
    AgentStatusDetail,
    MetricType,
)

# Workflow history models (0.2c)
from shared.database.models.workflow_history import (
    WorkflowExecution,
    WorkflowStep,
    WorkflowArtifact,
    # Enums
```

```
    WorkflowType,
    WorkflowStatus,
    StepStatus,
    ArtifactType,
)

# Resource schema models (0.2d) - NEW!
from shared.database.models.resource_schema import (
    ResourceMetric,
    ScalingEvent,
    # Enums
    ResourceType,
    ScalingEventType,
)

__all__ = [
    # Core models
    "Customer",
    "Agent",
    "Event",
    "Recommendation",
    "Approval",
    "Optimization",
    # Core enums
    "CustomerStatus",
    "AgentType",
    "AgentStatus",
    "EventType",
    "EventSeverity",
    "RecommendationType",
    "ApprovalStatus",
    "OptimizationStatus",
    # Agent state models
    "AgentConfig",
    "AgentState",
    "AgentCapability",
    "AgentMetric",
    # Agent state enums
    "ConfigType",
    "AgentStatusDetail",
    "MetricType",
```

```
    # Workflow history models
    "WorkflowExecution",
    "WorkflowStep",
    "WorkflowArtifact",
    # Workflow history enums
    "WorkflowType",
    "WorkflowStatus",
    "StepStatus",
    "ArtifactType",
    # Resource schema models - NEW!
    "ResourceMetric",
    "ScalingEvent",
    # Resource schema enums - NEW!
    "ResourceType",
    "ScalingEventType",
]
```

---

# 📁 FILE 4: Alembic Migration

**Location:** `~/optiinfra/shared/database/migrations/versions/004_resource_schema.py`

python

```
"""Create resource schema tables

Revision ID: 004_resource_schema
Revises: 003_workflow_history
Create Date: 2025-10-20 16:00:00.000000

"""
from alembic import op
import sqlalchemy as sa
from sqlalchemy.dialects import postgresql

# revision identifiers, used by Alembic.
revision = '004_resource_schema'
down_revision = '003_workflow_history'
branch_labels = None
depends_on = None


def upgrade() -> None:
    """Create resource schema tables"""

    # Create ENUM types
    resource_type_enum = postgresql.ENUM(
        'gpu',
        'cpu',
        'memory',
        'disk',
        'network',
        name='resource_type',
        create_type=True
    )
    resource_type_enum.create(op.get_bind(), checkfirst=True)

    scaling_event_type_enum = postgresql.ENUM(
        'scale_up',
        'scale_down',
        'auto_scale_triggered',
        'manual_scale',
        'scale_cancelled',
        name='scaling_event_type',
        create_type=True
```

```
)
scaling_event_type_enum.create(op.get_bind(), checkfirst=True)

# Create resource_metrics table
op.create_table(
    'resource_metrics',
    sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True),
    sa.Column('agent_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('customer_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('instance_id', sa.String(255), nullable=False),
    sa.Column('resource_type', resource_type_enum, nullable=False),
    sa.Column('metric_name', sa.String(100), nullable=False),
    sa.Column('metric_value', sa.Float(), nullable=False),
    sa.Column('unit', sa.String(50), nullable=False),
    sa.Column('timestamp', sa.DateTime(), nullable=False),
    sa.Column('metadata', postgresql.JSONB(), nullable=False, server_default='{}'),
    sa.Column('created_at', sa.DateTime(), nullable=False),
    sa.ForeignKeyConstraint(['agent_id'], ['agents.id'], ondelete='CASCADE'),
    sa.ForeignKeyConstraint(['customer_id'], ['customers.id'], ondelete='CASCADE'),
)

# Create indexes for resource_metrics
op.create_index('ix_resource_metrics_id', 'resource_metrics', ['id'])
op.create_index('ix_resource_metrics_agent_id', 'resource_metrics', ['agent_id'])
op.create_index('ix_resource_metrics_customer_id', 'resource_metrics', ['customer_id'])
op.create_index('ix_resource_metrics_instance_id', 'resource_metrics', ['instance_id'])
op.create_index('ix_resource_metrics_resource_type', 'resource_metrics', ['resource_type'])
op.create_index('ix_resource_metrics_metric_name', 'resource_metrics', ['metric_name'])
op.create_index('ix_resource_metrics_timestamp', 'resource_metrics', ['timestamp'])
op.create_index('ix_resource_metrics_agent_instance', 'resource_metrics', ['agent_id', 'instance_id'])
op.create_index('ix_resource_metrics_customer_type', 'resource_metrics', ['customer_id', 'resource_type'])
op.create_index('ix_resource_metrics_instance_timestamp', 'resource_metrics', ['instance_id', 'timestamp'])
op.create_index('ix_resource_metrics_type_name', 'resource_metrics', ['resource_type', 'metric_name'])

# Create scaling_events table
op.create_table(
    'scaling_events',
    sa.Column('id', postgresql.UUID(as_uuid=True), primary_key=True),
    sa.Column('agent_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('customer_id', postgresql.UUID(as_uuid=True), nullable=False),
    sa.Column('workflow_execution_id', postgresql.UUID(as_uuid=True), nullable=True),
```

```
        sa.Column('event_type', scaling_event_type_enum, nullable=False),
        sa.Column('trigger_reason', sa.String(500), nullable=False),
        sa.Column('before_state', postgresql.JSONB(), nullable=False),
        sa.Column('after_state', postgresql.JSONB(), nullable=False),
        sa.Column('success', sa.Boolean(), nullable=False, server_default='false'),
        sa.Column('error_details', postgresql.JSONB(), nullable=True),
        sa.Column('executed_at', sa.DateTime(), nullable=False),
        sa.Column('completed_at', sa.DateTime(), nullable=True),
        sa.Column('metadata', postgresql.JSONB(), nullable=False, server_default='{}'),
        sa.Column('created_at', sa.DateTime(), nullable=False),
        sa.Column('updated_at', sa.DateTime(), nullable=False),
        sa.ForeignKeyConstraint(['agent_id'], ['agents.id'], ondelete='CASCADE'),
        sa.ForeignKeyConstraint(['customer_id'], ['customers.id'], ondelete='CASCADE'),
        sa.ForeignKeyConstraint(['workflow_execution_id'], ['workflow_executions.id'], ondelete='SET NULL'),
    )

    # Create indexes for scaling_events
    op.create_index('ix_scaling_events_id', 'scaling_events', ['id'])
    op.create_index('ix_scaling_events_agent_id', 'scaling_events', ['agent_id'])
    op.create_index('ix_scaling_events_customer_id', 'scaling_events', ['customer_id'])
    op.create_index('ix_scaling_events_workflow_execution_id', 'scaling_events', ['workflow_execution_id'])
    op.create_index('ix_scaling_events_event_type', 'scaling_events', ['event_type'])
    op.create_index('ix_scaling_events_success', 'scaling_events', ['success'])
    op.create_index('ix_scaling_events_executed_at', 'scaling_events', ['executed_at'])
    op.create_index('ix_scaling_events_agent_type', 'scaling_events', ['agent_id', 'event_type'])
    op.create_index('ix_scaling_events_customer_success', 'scaling_events', ['customer_id', 'success'])
    op.create_index('ix_scaling_events_workflow', 'scaling_events', ['workflow_execution_id'])


def downgrade() -> None:
    """Drop resource schema tables"""

    # Drop scaling_events table and indexes
    op.drop_index('ix_scaling_events_workflow', 'scaling_events')
    op.drop_index('ix_scaling_events_customer_success', 'scaling_events')
    op.drop_index('ix_scaling_events_agent_type', 'scaling_events')
    op.drop_index('ix_scaling_events_executed_at', 'scaling_events')
    op.drop_index('ix_scaling_events_success', 'scaling_events')
    op.drop_index('ix_scaling_events_event_type', 'scaling_events')
    op.drop_index('ix_scaling_events_workflow_execution_id', 'scaling_events')
    op.drop_index('ix_scaling_events_customer_id', 'scaling_events')
```

```
    op.drop_index('ix_scaling_events_agent_id', 'scaling_events')
    op.drop_index('ix_scaling_events_id', 'scaling_events')
    op.drop_table('scaling_events')

    # Drop resource_metrics table and indexes
    op.drop_index('ix_resource_metrics_type_name', 'resource_metrics')
    op.drop_index('ix_resource_metrics_instance_timestamp', 'resource_metrics')
    op.drop_index('ix_resource_metrics_customer_type', 'resource_metrics')
    op.drop_index('ix_resource_metrics_agent_instance', 'resource_metrics')
    op.drop_index('ix_resource_metrics_timestamp', 'resource_metrics')
    op.drop_index('ix_resource_metrics_metric_name', 'resource_metrics')
    op.drop_index('ix_resource_metrics_resource_type', 'resource_metrics')
    op.drop_index('ix_resource_metrics_instance_id', 'resource_metrics')
    op.drop_index('ix_resource_metrics_customer_id', 'resource_metrics')
    op.drop_index('ix_resource_metrics_agent_id', 'resource_metrics')
    op.drop_index('ix_resource_metrics_id', 'resource_metrics')
    op.drop_table('resource_metrics')

    # Drop ENUM types
    sa.Enum(name='scaling_event_type').drop(op.get_bind(), checkfirst=True)
    sa.Enum(name='resource_type').drop(op.get_bind(), checkfirst=True)
```

---

# 🚀 EXECUTION STEPS

### Step 1: Create Models File

bash

```
cd ~/optiinfra/shared/database/models

# Create resource_schema.py
cat > resource_schema.py << 'EOF'
[Copy the entire resource_schema.py content from FILE 1 above]
EOF

# Verify file created
ls -lh resource_schema.py
# Expected: ~450 lines
```

## Step 2: Update Core Models

bash

```
cd ~/optiinfra/shared/database/models

# Open core.py and add relationships
nano core.py

# Add to Agent class:
#   resource_metrics = relationship(...)
#   scaling_events = relationship(...)
#
# Add to Customer class:
#   resource_metrics = relationship(...)
#   scaling_events = relationship(...)
```

## Step 3: Update Model Imports

bash

```
cd ~/optiinfra/shared/database/models

# Update __init__.py with resource models
cat > __init__.py << 'EOF'
[Copy the entire __init__.py content from FILE 3 above]
EOF
```

## Step 4: Create Migration File

bash

```
cd ~/optiinfra/shared/database/migrations/versions

# Create migration file
cat > 004_resource_schema.py << 'EOF'
[Copy the entire migration content from FILE 4 above]
EOF

# Verify file created
ls -lh 004_resource_schema.py
# Expected: ~180 lines
```

## Step 5: Test Imports

bash

```
cd ~/optiinfra

# Test if new models can be imported
python << 'EOF'
from shared.database.models.resource_schema import (
    ResourceMetric,
    ScalingEvent,
    ResourceType,
    ScalingEventType
)
print('✅ All resource models import successfully')

# Test enums
print(f'✅ ResourceType has {len(ResourceType)} values')
print(f'✅ ScalingEventType has {len(ScalingEventType)} values')
EOF

# Expected output:
# ✅ All resource models import successfully
# ✅ ResourceType has 5 values
# ✅ ScalingEventType has 5 values
```

### Step 6: Run Alembic Migration

bash

```
cd ~/optiinfra/shared/database

# Check current migration state
alembic current
# Expected: 003_workflow_history (current)

# Run the new migration
alembic upgrade head

# Expected output:
# INFO  [alembic.runtime.migration] Running upgrade 003_workflow_history -> 004_resource_schema, Create resource
```

## Step 7: Verify Tables Created

bash

```
# Connect to PostgreSQL
psql postgresql://optiinfra:password@localhost:5432/optiinfra

# List all tables (should now be 15 tables)
\dt

# Expected output (13 from previous + 2 new):
#  resource_metrics
#  scaling_events
#  workflow_executions
#  workflow_steps
#  workflow_artifacts
#  agent_configs
#  agent_states
#  agent_capabilities
#  agent_metrics
#  customers
#  agents
#  events
#  recommendations
#  approvals
#  optimizations

# Describe resource_metrics table
\d resource_metrics

# Expected: Should show all columns, indexes, foreign keys

# Describe scaling_events table
\d scaling_events

# Expected: Should show all columns, foreign keys to agents, customers, workflows

# Check new enums were created
\dT+

# Expected: Should see resource_type, scaling_event_type
```

```
# Exit
\q
```

## Step 8: Verify Relationships Work

bash

```
cd ~/optiinfra

# Test relationships in Python
python << 'EOF'
from shared.database.models import Agent, Customer, ResourceMetric, ScalingEvent
from shared.database.session import SessionLocal

db = SessionLocal()

# Get a test agent
agent = db.query(Agent).first()
print(f"✅ Agent: {agent.name}")
print(f"   - Has resource_metrics relationship: {hasattr(agent, 'resource_metrics')}")
print(f"   - Has scaling_events relationship: {hasattr(agent, 'scaling_events')}")

# Get a test customer
customer = db.query(Customer).first()
print(f"✅ Customer: {customer.name}")
print(f"   - Has resource_metrics relationship: {hasattr(customer, 'resource_metrics')}")
print(f"   - Has scaling_events relationship: {hasattr(customer, 'scaling_events')}")

# Check models
print(f"✅ ResourceMetric model loaded")
print(f"   - Has agent relationship: {hasattr(ResourceMetric, 'agent')}")
print(f"   - Has customer relationship: {hasattr(ResourceMetric, 'customer')}")

print(f"✅ ScalingEvent model loaded")
print(f"   - Has agent relationship: {hasattr(ScalingEvent, 'agent')}")
print(f"   - Has customer relationship: {hasattr(ScalingEvent, 'customer')}")
print(f"   - Has workflow relationship: {hasattr(ScalingEvent, 'workflow')}")

db.close()
print("\n✅ All relationship verifications passed!")
EOF

# Expected output similar to workflow verification
```

---

# 🎯 PART 1 SUMMARY

## Files Created/Modified:

✅ **shared/database/models/resource_schema.py** - 2 new models + 2 enums (~450 lines)
✅ **shared/database/models/core.py** - Modified (added 4 relationships)
✅ **shared/database/models/init.py** - Modified (exports)
✅ **shared/database/migrations/versions/004_resource_schema.py** - Migration (~180 lines)

## What Works Now:

✅ Can run Alembic migration
✅ 2 new tables created in PostgreSQL
✅ All foreign keys working (cascades configured)
✅ All indexes created (20+ indexes)
✅ Agent-resource relationship working
✅ Customer-resource relationship working
✅ Scaling events link to workflows

## Database Status:

Total Tables: 15 (88% PostgreSQL schema complete!)
├── Core (0.2a): 6 tables ✅
├── Agent State (0.2b): 4 tables ✅
├── Workflow History (0.2c): 3 tables ✅
└── Resource Schema (0.2d): 2 tables ✅

Total Enums: 17
Total Indexes: 80+
Total Relationships: 29+

## What's in Part 2:

- Seed data (resource metrics, scaling events for test scenarios)
- Test fixtures (sample metrics, scaling events)
- Test cases (12-15 comprehensive tests)
- Integration tests (resource tracking, scaling flow)
- All validation commands
- Troubleshooting guide
- Success criteria checklist
- Git commit instructions

---

# ⏭️ NEXT STEP

**Download PART 2** to get:

- Complete seed data implementation (realistic GPU/CPU metrics)
- Full test suite (12-15 tests covering all scenarios)
- Step-by-step validation
- Troubleshooting for common issues
- Final checklist

**File name:** `FOUNDATION-0.2d-Resource-Schema-PART2-Testing.md`

---

# 📥 HOW TO DOWNLOAD THIS FILE

1. Click the **"Copy"** dropdown button at the top of this artifact
2. Select **"Download as txt"**
3. Save as: `FOUNDATION-0.2d-Resource-Schema-PART1-Code.md`

---

# ✅ VERIFICATION CHECKLIST (Part 1)

Before moving to Part 2, verify:

- ☐ `resource_schema.py` created with 2 models
- ☐ `core.py` updated with 4 new relationships
- ☐ `__init__.py` exports all resource models
- ☐ Migration file `004_resource_schema.py` created
- ☐ Migration runs successfully (`alembic upgrade head`)
- ☐ 2 new tables exist in PostgreSQL
- ☐ 2 new enum types created
- ☐ All indexes created
- ☐ Python imports work (no errors)
- ☐ Relationships work (verified in Python)

**If all checked, you're ready for PART 2!** ✅

---

✅ **PART 1 COMPLETE! Ready for PART 2?** 🚀 import all new models in Python
✅ Can